**Java Practical Assessment-3**

**THEORY QUESTIONS:**

**Q1. What are the major features in different versions of Spring Framework?**

**Ans:**

**1.Version Spring 2.5**

**2.Spring 3.0**

**3.Spring 4.0**

Feature This version was released in 2007. It was the first version which supported annotations. This version was released in 2009. It made full-fledged use of improvements in Java5 and also provided support to JEE6. This version was released in 2013. This was the first version to provide full support to Java 8.

**Q2. What is a Spring Framework?**

**Ans:** Spring is a powerful open source, application framework created to reduce the complexity of enterprise application development. It is light-weighted and loosely coupled. It has layered architecture, which allows you to select the components to use, while also providing a cohesive framework for J2EE application development. Spring framework is also called the framework of frameworks as it provides support to various other frameworks such as Struts, Hibernate, EJB, JSF etc.

**Q3. List the advantages of Spring Framework**

**Ans:**

**1.Spring is non-invasive:** That means you no need to implements any interface or inherit any class from spring to your classes, so whenever you want to change from spring to any other technology then you no need to change the logics of your class.

**2.Spring is light weight:** Spring is vast framework so spring people divide the whole spring in to different modules, they are designed in such a way that no module is dependent to other module, excep Spring core module, so according to your requirement you can learn a particular module, you no need to learn whole total framework.

**3.End to end Development:** Spring supports all aspects of application development, Business aspects, persistence aspects, etc, so we can develop a complete application using spring.

**4.Spring supports All types of application development:** We can develop any type of applications using spring, eg: Core java, web Application, Distributed application, Enterprise application.

**5.Spring is versatile:** We can integrate any technologies with spring, so we can say spring is versatile.

**6.Spring supports dependency injection:** The dependency between classes are managed by spring.

**Q4. What are the different features of Spring Framework?**

**Ans:** Following are some of the major features of Spring Framework:

**1.Lightweight:** Spring is lightweight when it comes to size and transparency. Inversion of control (IOC): The objects give their dependencies instead of creating or looking for dependent objects. This is called Inversion of Control.

**2.Aspect oriented Programming (AOP):** Aspect oriented programming in Spring supports cohesive development by separating application business logic from system services.

**3.Container:** Spring Framework creates and manages the life cycle and configuration of the application objects. MVC Framework: Spring Framework’s MVC web application framework is highly configurable. Other frameworks can also be used easily instead of Spring MVC Framework.

**4.Transaction Management:** Generic abstraction layer for transaction management is provided by the Spring Framework.

5.Spring’s transaction support can be also used in container less environments.

6.**JDBC Exception Handling:** The JDBC abstraction layer of the Spring offers an exception hierarchy, which simplifies the error handling strategy.

**Q5. How many modules are there in Spring Framework and what are they?**

**Ans:** There are around 20 modules which are generalized into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation and Test.

**1.Spring Core Container** – This layer is basically the core of Spring Framework. It contains the following modules: Spring Core Spring Bean SPEL (Spring Expression Language) Spring Context

**2.Data Access/Integration** – This layer provides support to interact with the database.

It contains the following modules:

1.JDBC (Java Database Connectivity)

2.ORM (Object Relational Mapping)

3.OXM (Object XML Mappers)

4.JMS (Java Messaging Service) Transactions.

**3.Web** – This layer provides support to create web application. It contains the following modules

**4.Web Web** – MVC Web – Socket Web – Portlet.

Aspect Oriented Programming (AOP) – In this layer you can use Advices, Pointcuts etc., to decouple the code. Instrumentation – This layer provides support to class instrumentation and class loader implementations. Test -This layer provides support to testing with JUnit and TestNG.

Few Miscellaneous modules are given below:

1.Messaging – This module provides support for STOMP. It also supports an annotation programming model that is used for routing and processing STOMP messages from WebSocket clients.

2.Aspects – This module provides support to integration with AspectJ.

**Q6. What is a Spring configuration file?**

**Ans:** A Spring configuration file is an XML file. This file mainly contains the classes information. It describes how those classes are configured as well as introduced to each other. The XML configuration files, however, are verbose and moreclean. If it’s not planned and written correctly, it becomes very difficult to manage in big projects.

**Q7. What are the different components of a Spring application?**

**Ans:** A Spring application, generally consists of following components:

1.**Interface**: It defines the functions.

2.**Bean class**: It contains properties, its setter and getter methods, functions etc.

3.**Spring Aspect Oriented Programming (AOP):** Provides the functionality of cross-cutting concerns.

4.**Bean Configuration File**: Contains the information of classes and how to configure then.

5.**User program**: It uses the function.

**Q8. What are the various ways of using Spring Framework?**

**Ans:** Spring Framework can be used in various ways. They are listed as follows:

1.As a Full-fledged Spring web application.

2.Asathirdpartywebframework, usingSpringFrameworksmiddle-tier.

3.For remote usage.

4.As Enterprise Java Bean which can wrap existing POJOs(PlainOldJava Objects).

**Q9. What is Spring IOC Container?**

**Ans:** At the core of the Spring Framework, lies the Spring container, the container creates the object, wires them together, configures them and manages their complete life cycle. The Spring container makes use of Dependency Injection to manage the components that make up an application. The container receives instructions for which objects to instantiate, configure, and assemble by reading the configuration metadata provided. This metadata can be provided either by XML, Java annotations or Java code.

**Q10. What do you mean by Dependency Injection?**

**Ans:** In Dependency Injection, you do not have to create your objects but have to describe how they should be created. You don’t connect your components and services together in the code directly, but describe which services are needed by which components in the configuration file. The IoC container will wire them up together.

**PRACTICAL QUESTIONS:**

**Q1. Please create one Spring core project with Employee and Laptop bean injected using setter and constructor both?**

**Ans.** **Main Program:**

Import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApplication {

public static void main (String[] args) {

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("data.xml");

Employee emp = (Employee)context.getBean("emp1");

Laptop laptop = (Laptop)context.getBean("laptop1");

System.out.println(emp);

}

}

**Employee Class:**

public class Employee {

private String employeeName;

private int phoneNumber, id;

public String employeeName() {

return employeeName;

}

public void setemployeeName(String employeeName) {

this.employeeName = employeeName;

}

public int getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(int phoneNumber) {

this.phoneNumber = phoneNumber;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

*@Override*

public String toString() {

return "Employee [employeeName=" + employeeName + ", phoneNumber=" + phoneNumber + ", id=" + id + "]";

}

}

**Laptop Class:**

public class Laptop {

private long laptopID;

private String laptopName;

public long getlaptopID() {

return laptopID;

}

public void setlaptopID(long laptopID) {

this.laptopID = laptopID;

}

public String getlaptopName() {

return laptopName;

}

public void setlaptopName(String laptopName) {

this.laptopName = laptopName;

}

public Employee(long laptopID, String laptopName) {

super();

this.laptopID = laptopID;

this.laptopName = laptopName;

}

*@Override*

public String toString() {

return "Product [laptopID=" + laptopID + ", laptopName=" + laptopName + "]";

}

}

**Data.xml file:**

<?xml version="1.0" encoding="UTF-8"?>

<!-- XML rules verification done by xml name space -->

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<!-- Extensible Markup Language Name Space -->

<!-- id is also name of bean and name is consider as alias name -->

<bean class="Laptop" id ="laptop" >

<property name="employeeName" value="Venu"></property>

<property name="phoneNumber" value = "837478"></property>

<property name="id" value = "01"></property>

</bean>

<!-- Injection using setters -->

<bean class = "Employee" id = "emp" >

<property name = "laptop" ref = "laptop"></property>

</bean>

<!-- Injection using constructors -->

<bean class = "Employee" id = "emp" >

<constructor-arg value="laptop" ></constructor-arg>

</bean>

</beans>

**Q2. Please create One to One relationship using hibernate?**

Ans.

**Employee class:**

import java.io.Serializable;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import javax.persistence.Table;

@Entity

@Table(name="Employee")

public class Employee implements Serializable{

/\*\*

\*

\*/

private static final long serialVersionUID = -7409929725102932771L;

@Id // Creating primary key column

@GeneratedValue(strategy = GenerationType.AUTO) // Automatic generating primary key for employee id here

@Column(name="EmployeeID") // Naming column as specified name

private int empID;

@Column(name="EmployeeName")

private String empName;

@Column(name="EmployeeAddress")

private String empAddress;

@Column(name="EmployeeContact")

private long empContact;

@OneToOne(cascade = CascadeType.ALL) // perform Operation on the child class

private Laptop laptop;

public int getEmpID() {

return empID;

}

public void setEmpID(int empID) {

this.empID = empID;

}

public String getEmpName() {

return empName;

}

public void setEmpName(String empName) {

this.empName = empName;

}

public String getEmpAddress() {

return empAddress;

}

public void setEmpAddress(String empAddress) {

this.empAddress = empAddress;

}

public long getEmpContact() {

return empContact;

}

public void setEmpContact(long empContact) {

this.empContact = empContact;

}

public Laptop getLaptop() {

return laptop;

}

public void setLaptop(Laptop laptop) {

this.laptop = laptop;

}

}

**Laptop class:**

import java.util.Objects;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

//@Audited

public class Laptop {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Integer laptopID;

//@Audited

private String modelNo;

public Integer getLaptopID() {

return laptopID;

}

public void setLaptopID(Integer laptopID) {

this.laptopID = laptopID;

}

public String getModelNo() {

return modelNo;

}

public void setModelNo(String modelNo) {

this.modelNo = modelNo;

}

@Override

public int hashCode() {

return Objects.hash(laptopID, modelNo);

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Laptop other = (Laptop) obj;

return Objects.equals(laptopID, other.laptopID) && Objects.equals(modelNo, other.modelNo);

}

}

**LaptopAdd Class:**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class LaptopAdd {

public static void main(String[] args) {

// TODO Auto-generated method stub

SessionFactory factory = new Configuration().configure().buildSessionFactory();

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

Laptop laptop = new Laptop();

laptop.setModelNo("Omen15");

// laptop.setLaptopID(15); // Not Needed as it's already auto generated.

session.save(laptop);

tx.commit();

factory.close();

}

}

**OneToOneRelationShip:**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class OneToOneRelationShip {

public static void main(String[] args) {

// TODO Auto-generated method stub

SessionFactory factory = new Configuration().configure().buildSessionFactory();

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

Laptop laptop=new Laptop();

laptop.setModelNo("Omen15");

Employee employee=new Employee();

employee.setEmpAddress("Visakhapatnam");

employee.setEmpContact(440633);

employee.setEmpName("Venu PUPPALA");

employee.setEmpID(803454);

employee.setLaptop(laptop);

session.save(laptop);

session.save(employee);

tx.commit();

factory.close();

}

}

**Hibernate.cfg.xml:**

<?xml version="1.0" encoding="UTF-8"?>

<!-- ~ Hibernate, Relational Persistence for Idiomatic Java ~ ~ License:

GNU Lesser General Public License (LGPL), version 2.1 or later. ~ See the

lgpl.txt file in the root directory or <http://www.gnu.org/licenses/lgpl-2.1.html>. -->

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.hbm2ddl.auto">update</property> <!-- Only for Development Env -->

<property name="hibernate.generate\_statistics">true</property>

<property name="connection.pool\_size">10</property>

<property name="show\_sql">true</property> <!-- Only for Development Env -->

<property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatedemo</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">Mysql#803454</property>

<mapping class="Employee" />

<mapping class="Laptop" />

<!-- <mapping class="Phone" />-->

</session-factory>

</hibernate-configuration>

**Q3. Please create one singleton and thread safe class example?**

Ans.

**Employee Class:**

import java.util.Objects;

public class Employee {

private long empID;

private String name, address, phone;

private boolean salaryPaid;

public Employee(long empID, String name, String address, String phone, boolean salaryPaid) {

super();

this.empID = empID;

this.name = name;

this.address = address;

this.phone = phone;

this.salaryPaid = salaryPaid;

}

public long getEmpID() {

return empID;

}

public void setEmpID(long empID) {

this.empID = empID;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

public boolean isSalaryPaid() {

return salaryPaid;

}

public void setSalaryPaid(boolean salaryPaid) {

this.salaryPaid = salaryPaid;

}

*@*Override

public int hashCode() {

return Objects.hash(address, empID, name, phone, salaryPaid);

}

*@*Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Employee other = (Employee) obj;

return Objects.equals(address, other.address) && empID == other.empID && Objects.equals(name, other.name) && Objects.equals(phone, other.phone) && salaryPaid == other.salaryPaid;

}

*@*Override

public String toString() {

return "Employee [empID=" + empID + ", name=" + name + ", address=" + address + ", phone=" + phone

+ ", salaryPaid=" + salaryPaid + ", getEmpID()=" + getEmpID() + ", getName()=" + getName()

+ ", getAddress()=" + getAddress() + ", getPhone()=" + getPhone() + ", isSalaryPaid()=" + isSalaryPaid()

+ ", hashCode()=" + hashCode() + ", getClass()=" + getClass() + ", toString()=" + super.toString()

+ "]";

}

}

**Main Application:**

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// TODO Auto-generated method stub

ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext("myString.xml");

Employee e=(Employee)applicationContext.getBean("emp123"); System.out.println(e);

}

}

**myString.xml:**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- bean definitions here -->

<bean class="Employee" name="emp123" scope="singleton">

<constructor-arg value="803454" ></constructor-arg>

<constructor-arg value="Venu Puppala" ></constructor-arg>

<constructor-arg value="Vizag" ></constructor-arg>

<constructor-arg value="8374784879" ></constructor-arg>

<constructor-arg value="false" ></constructor-arg>

</bean>

</beans>

**Q4. Please show the difference between singleton and prototype scope in spring?**

Ans. **Employee Class:**

package com.venup;

import java.util.Objects;

public class Employee {

private long eid;

private String name;

private Laptop laptop;

// public Employee(String name) {

// super();

// this.name = name;

// }

//

// public Employee(long eid) {

// super();

// this.eid = eid;

// }

public Employee(long eid, String name) {

super();

this.eid = eid;

this.name = name;

}

public long getEid() {

return eid;

}

public void setEid(long eid) {

this.eid = eid;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Laptop getLaptop() {

return laptop;

}

public void setLaptop(Laptop laptop) {

this.laptop = laptop;

}

// @Override

// public int hashCode() {

// return Objects.hash(eid, name);

// }

// @Override

// public boolean equals(Object obj) {

// if (this == obj) {

// return true;

// }

// if (obj == null) {

// return false;

// }

// if (getClass() != obj.getClass()) {

// return false;

// }

// Employee other = (Employee) obj;

// return eid == other.eid && Objects.equals(name, other.name);

// }

*@Override*

public String toString() {

return "Employee [eid=" + eid + ", name=" + name + ", laptop=" + laptop + "]";

}

}

**Laptop Class:**

package com.venup;

import java.util.Objects;

public class Laptop {

private long lid;

private String cpu;

private int ram;

public long getLid() {

return lid;

}

public void setLid(long lid) {

this.lid = lid;

}

public String getCpu() {

return cpu;

}

public void setCpu(String cpu) {

this.cpu = cpu;

}

public int getRam() {

return ram;

}

public void setRam(int ram) {

this.ram = ram;

}

// @Override

// public int hashCode() {

// return Objects.hash(cpu, lid, ram);

// }

// @Override

// public boolean equals(Object obj) {

// if (this == obj) {

// return true;

// }

// if (obj == null) {

// return false;

// }

// if (getClass() != obj.getClass()) {

// return false;

// }

// Laptop other = (Laptop) obj;

// return Objects.equals(cpu, other.cpu) && lid == other.lid && ram == other.ram;

// }

*@Override*

public String toString() {

return "laptop [lid=" + lid + ", cpu=" + cpu + ", ram=" + ram + ", getLid()=" + getLid() + ", getCpu()="

+ getCpu() + ", getRam()=" + getRam() + ", hashCode()=" + hashCode() + ", getClass()=" + getClass()

+ ", toString()=" + super.toString() + "]";

}

}

**myString.xml:**

<?xml version="1.0" encoding="UTF-8"?>

<!-- XML rules verification done by xml name space -->

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<!-- Extensible Markup Language Name Space -->

<!-- id is also name of bean and name is consider as alias name -->

<bean class="com.venup.Laptop" name="mylaptop" scope="singleton">

<!-- Dependency injection using Setter Methods -->

<property name="lid" value="100"></property>

<property name="cpu" value="Lenovo "></property>

<property name="ram" value="16"></property>

</bean>

<bean class="com.venup.Employee" id="emp" scope="prototype">

<!-- Dependency injection using constructor -->

<constructor-arg value="06" ></constructor-arg>

<constructor-arg value="Venu"></constructor-arg>

<property name="laptop" ref="mylaptop"></property>

</bean>

<bean class="com.venup.Employee" id="emp1" scope="prototype">

<!-- Dependency injection using constructor -->

<constructor-arg value="18" ></constructor-arg>

<constructor-arg value="chandra"></constructor-arg>

<property name="laptop" ref="mylaptop"></property>

</bean>

</beans>

**Q5. Please create One to Many bidirectional relationships with Spring Boot Data JPA?**  
Ans. Create a Spring Boot Project in Eclipse and take dependencies as Spring Web, Spring JPA and the data base you are using in my case it is MY SQL Driver.

**Application.properties:**

spring.datasource.url=jdbc:mysql://localhost:3306/hibernateonetomany?useSSL=false&serverTimezone=UTC&useLegacyDatetimeCode=false

spring.datasource.username=root

spring.datasource.password=Mysql#803454

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.generate-ddl=true

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

server.port=8086

**OneToManyAplication:**

package com.springboot\_hybernet\_one\_to\_many;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootHibernateOneToManyApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootHibernateOneToManyApplication.class, args);

}

}

**Employee Class:**

package com.springboot\_hybernet\_one\_to\_many.model;

import java.io.Serializable;

//import java.util.List;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

@Entity

public class Employee implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 7108602029108606198L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long eid;

private String name, address;

// @OneToMany(cascade = CascadeType.ALL) // To create a Table

@OneToMany(cascade = CascadeType.ALL, mappedBy = "employee") // To create a column

// private List<Phone> phones; // To allow duplicates

private Set<Phone> phones; // To not allow duplicates

public Long getEid() {

return eid;

}

public void setEid(Long eid) {

this.eid = eid;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public Set<Phone> getPhones() {

return phones;

}

public void setPhones(Set<Phone> phones) {

this.phones = phones;

}

// public List<Phone> getPhones() {

// return phones;

// }

//

// public void setPhones(List<Phone> phones) {

// this.phones = phones;

// }

}

**Phone Class:**

package com.springboot\_hybernet\_one\_to\_many.model;

import javax.persistence.Entity;

import javax.persistence.EnumType;

import javax.persistence.Enumerated;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.ManyToOne;

@Entity

public class Phone {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long pid;

private String comment, phoneNumber;

@ManyToOne

private Employee employee;

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

@Enumerated (EnumType.STRING)

private PHONE\_TYPE phone\_TYPE;

public Long getPid() {

return pid;

}

public void setPid(Long pid) {

this.pid = pid;

}

public String getComment() {

return comment;

}

public void setComment(String comment) {

this.comment = comment;

}

public PHONE\_TYPE getPhone\_TYPE() {

return phone\_TYPE;

}

public void setPhone\_TYPE(PHONE\_TYPE phone\_TYPE) {

this.phone\_TYPE = phone\_TYPE;

}

}

**Phone\_Type enum:**

package com.springboot\_hybernet\_one\_to\_many.model;

public enum PHONE\_TYPE {

*HOME*, *OFFICE*, *EMERGENCY*, *PERMANENT*;

}

**EmployeeRepository Interface:**

package com.springboot\_hybernet\_one\_to\_many.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.springboot\_hybernet\_one\_to\_many.model.Employee;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long>{

}

**PhoneRepository Interface:**

package com.springboot\_hybernet\_one\_to\_many.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.springboot\_hybernet\_one\_to\_many.model.Phone;

@Repository

public interface PhoneRepository extends JpaRepository<Phone, Long>{

}

**EmployeeRestController Class:**

package com.springboot\_hybernet\_one\_to\_many.restcontroller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.springboot\_hybernet\_one\_to\_many.model.Employee;

import com.springboot\_hybernet\_one\_to\_many.repository.EmployeeRepository;

@RestController

@RequestMapping("/employee")

public class EmployeeRestController {

@Autowired

private EmployeeRepository empRepo;

@PostMapping("/")

public Employee addEmployee(@RequestBody Employee emp) {

return empRepo.save(emp);

}

@GetMapping("/")

public List<Employee> listEmployee() {

return empRepo.findAll();

}

@PutMapping("/")

public Employee updateEmployee(@RequestBody Employee emp) {

return empRepo.save(emp);

}

@DeleteMapping("/")

public String deleteEmployee(@RequestBody Employee emp) {

empRepo.delete(emp);

return "Deleted Employee Data";

}

}

**PhoneRestController Class:**

package com.springboot\_hybernet\_one\_to\_many.restcontroller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.springboot\_hybernet\_one\_to\_many.model.Phone;

import com.springboot\_hybernet\_one\_to\_many.repository.PhoneRepository;

@RestController

@RequestMapping("/phone")

public class PhoneRestController {

@Autowired

private PhoneRepository phnRepo;

@PostMapping("/")

public Phone addPhone(@RequestBody Phone phone) {

return phnRepo.save(phone);

}

@GetMapping("/")

public List<Phone> listPhone() {

return phnRepo.findAll();

}

@PutMapping("/")

public Phone updatePhone(@RequestBody Phone phone) {

return phnRepo.save(phone);

}

@DeleteMapping("/")

public String deletePhone(@RequestBody Phone ph) {

phnRepo.delete(ph);

return "Phone Data Deleted";

}

}